**(9) WPF 컨트롤 다루기**

**Button 컨트롤 동작**  
지금까지 의존성 속성과 이벤트 라우틴에 대해서 살펴보았고 이 내용을 통해서 WPF 컨트롤을 보다 쉽게 이해할 수 있을 것이다. 그럼 버튼(Button) 컨트롤부터 살펴보도록 하겠다. 본능적으로 우리는 이미 버튼 컨트롤의 역할에 대해서 알고 있다. 버튼 컨트롤들은 마우스를 눌렀을 때나 키보드에서 엔터키를 눌렀을 때 그리고 버튼에 포커스가 왔을 때 각각의 UI를 가지고 있다. WPF에서는 ButtonBase라는 클래스가 존재하고 이 클래스는 Button, RepeatButton, ToggleButton와 같은 컨트롤들로 파생된다.   
  
**ButtonBase 클래스**  
어떤 부모 클래스들과 마찬가지로 ButtonBase 클래스는 파생한 컨트롤들에게 여러 가지 모양의 인터페이스를 제공하고 있다 또한 Click 이벤트를 정의하고 있고 IsPressed라는 속성 또한 정의하고 있다. 이 속성은 버튼이 파생된 버튼이 눌려진 적이 있는지를 검사하기 위해서 사용한다. 이 밖에도 다음 [표29-2]에서는 ButtonBase에서 제공하고 있는 멤버들에 대해서 설명하고 있다.

|  |  |
| --- | --- |
| 속성 | 뜻 |
| ClickMode | 이 속성은 Click 이벤트를 언제 발생시킬지를 결정하고 Enum형으로 선언되어있다. |
| Command | 앞장에서 설명했던 것처럼 많은 UI요소들은 특정 명령을 가질 수 있고 이 명령은 Command 속성을 이용해서 할당하게 된다. |
| CommandParameter | 이 속성은 Command 속성에 지정된 파라메터를 전달하기 위한 속성이다. |
| CommandTarget | 이 속성은 Command 속성에 설정한 명령을 받을 대상을 지정한다. |

[표29-2] ButtonBase 클래스의 멤버들  
  
명령과 관련된 멤버들 중에 가장 흥미로운 멤버는 바로 ClickMode 속성이다. 이 속성은 버튼이 클릭될 시점을 다르게 지정할 수 있다. 이 속성은 System.Windows.Controls.ClickMode에서 제공되는 다음과 같은 값들 중에 하나를 지정할 수 있다.

|  |
| --- |
| public enum ClickMode {   Release,   Press,   Hover } |

예를 들어 다음과 같이 ClickMode 속성에 ClickMode.Hover를 지정해 주었다고 가정해보자.

|  |
| --- |
| <Button Name = "bntHoverClick" ClickMode = "Hover" Click ="btnHoverClick\_Click"/> |

이 경우 클릭 이벤트는 마우스 커서를 버튼의 어디든 가져가게 되면 발생하게 된다. 대부분의 경우 푸쉬(push)버튼을 이용하게 될 것이지만 Hover 모드의 경우 커스텀 스타일이나 템플릿, 애니메이션과 같은 동작을 만들고자 할 경우 유용할 수 있다.   
  
  
**Button 컨트롤**  
Button 클래스는 IsCancel, IsDefault라는 재미있는 두 가지 속성을 가지고 있다. 이 속성은 우리가 OK와 Cancel이 있는 대화 상자를 만들 때 매우 유용하게 사용할 수 있다. IsCancel을 true로 설정하면 버튼은 사용자가 ESC 키를 누른 것처럼 동작된다. 만약 IsDefault를 true로 설정하면 버튼은 인위적으로 엔터키를 누른 것처럼 동작하게 된다. 그럼 XAML로 다음과 같은 버튼이 있다고 가정해보자.

|  |
| --- |
| <!-- Assume these are defined within a <StackPanel> of a Window type --> <Button Name ="btnOK" IsDefault = "true" Click ="btnOK\_Click" Content = "OK"/> <Button Name ="btnCancel" IsCancel= "true" Click ="btnCancel\_Click" Content = "Cancel"/> |

만약 이 코드를 작성한 후에 실행해 보면 Enter키와 ESC키를 누른 것처럼 동작되는 것을 볼 수 있을 것이다. 만약 다른 Window의 다른 UI 요소에 포커스가 있다면 거기에서 또한 키가 동작되는 것을 볼 수 있을 것이다.  
  
  
**ToggleButton 컨트롤**  
  
System.Windows.Controls.Primitives네임스페이스에 정의 되어 있는 ToggleButton은 기본적으로는 Button 컨트롤과 동일한 UI를 가지고 있다. 하지만 이 버튼은 클릭되었을 때의 눌려진 상태를 유지할 수 있는 기능을 가지고 있다. ToggleButto은 IsChecked 속성을 제공하고 있고 사용자가 UI를 클릭했는지에 따라서 true와 false값을 설정하게 된다. 또한 ToggleButton은 상태 변화를 감지할 수 있는 Checked와 UnChecked 라는 두 가지 이벤트를 제공하고 있다. 다음 XAML은 토글의 이벤트를 어떻게 받아서 처리하는지의 예를 보여주고 있다.

|  |
| --- |
| <!-- Yes/No ToggleButton --> <ToggleButton Name ="toggleOnOffButton" Checked ="toggleOnOffButton\_Checked" Unchecked ="toggleOnOffButton\_Unchecked"> Off! </ToggleButton > |

이 이벤트는 간단하게 Content의 메시지를 변경해주고 있다.

|  |
| --- |
| protected void toggleOnOffButton\_Checked(object sender, RoutedEventArgs e) {     toggleOnOffButton.Content = "On!"; } protected void toggleOnOffButton\_Unchecked(object sender, RoutedEventArgs e) {     toggleOnOffButton.Content = "Off!"; } |

만약 코드에서 각각의 이벤트를 하나의 이벤트로 통합하고 싶다면 XAML로 정의된 Checked, Unchecked 이벤트를 toggleOnOffButtonPressed 하나로 생성해서 IsChecked 속성을 이용해서 작업할 수 있다.

|  |
| --- |
| protected void toggleOnOffButtonPressed(object sender, RoutedEventArgs e) {     if (toggleOnOffButton.IsChecked == false )         toggleOnOffButton.Content = "Off!";     else         toggleOnOffButton.Content = "On!"; } |

마지막으로 ToggleButton은 IsThreeState 속성과 Indeterminate 이벤트를 이용해서 세 가지 상태에 대한 정보를 알 수 있다. 즉, 체크된 상태와 체크가 해제된 상태 그리고 둘 다 아닌 상태가 존재한다는 것이다. 이러한 버튼이 조금 이상해 느껴질 수도 있지만 체크박스의 기능을 가지고 있는 버튼으로 생각하면 충분히 유용할 것이다.

|  |
| --- |
| System.Windows.Controls.Primitives라는 네임스페이스에 포함되어 있는 이 버튼은 추가적인 커스텀마이징 없이 바로 사용하기에는 약간의 무리가 있다. |

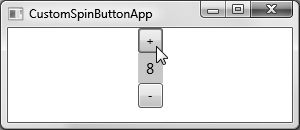
**RepeatButton 컨트롤**  
마지막으로 ButtonBase 파생된 RepeatButton컨트롤 또한 System.Windows.Controls.Primitives 네임스페이스에 포함되어 있다. 이 컨트롤은 기본적으로 일반 Button 컨트롤과 다르지 않다. 하지만 만약 사용자가 버튼을 누르고 있을 경우 연속적으로 Click 이벤트를 발생시킬 수 있는 기능을 가지고 있다. Click 이벤트가 발생되는 빈도는 Delay와 Interval 속성을 이용해서 밀리세컨드로 할당할 수 있다.   
  
실제로 RepeatButton 컨트롤은 ToggleButton 처럼 그렇게 유용하지 않다. 하지만 사용자 인터페이스를 커스텀마이싱한다면 어느 정도 유용할 것이다. 윈도우 폼과는 다르게 WPF는 위 아래 버튼을 이용해서 숫자를 설정하는 스핀(spin)버튼 컨트롤을 지원하지 않고 있다. 스핀 버튼을 구현하고 싶다면 RepeatButton을 커스텀마이징해야할 것이다.  
  
그럼 RepeatButton을 다루어 보도록 하자. 먼저 CustomSpinButtonApp라는 WPF 응용 프로그램 프로젝트를 생성해 보도록 하자. 처음에 <Grid>로 정의되어 있는 판낼을 <StackPanel>로 변경해 보도록 하자.

|  |
| --- |
| private void btnPurchaseOptions\_Click(object sender, RoutedEventArgs e) {     MessageBox.Show("Button has been clicked"); } |

   
RepeatButton 클래스는 각각의 Click 이벤트를 작성해 주어야 한다는 것을 알아두자. 그럼 C#에서 값이 증가하고 감소하는 값을 Label에 보여주도록 로직을 작성해 보도록 하자. 만약 여기서 추가적으로 최대값과 최소값을 설정해주고 싶다면 추가해 주어도 상관없다. 필자는 이 로직은 생략하였다.

|  |
| --- |
| <Window x:Class="CustomSpinButtonApp.MainWindow" xmlns="[**http://schemas.microsoft.com/winfx/2006/xaml/presentation**](http://schemas.microsoft.com/winfx/2006/xaml/presentation)" xmlns:x="[**http://schemas.microsoft.com/winfx/2006/xaml**](http://schemas.microsoft.com/winfx/2006/xaml)" Title="CustomSpinButtonApp" Height="300" Width="300">      <StackPanel>           <!-- 'Up' 버튼 -->           <RepeatButton Height ="25" Width = "25" Name ="repeatAddValueButton"                Delay ="200" Interval ="1" Click ="repeatAddValueButton\_Click"                Content = "+"/>           <!-- 현재 값 설정 -->           <Label Name ="lblCurrentValue" Background ="LightGray"                Height ="30" Width = "25"VerticalContentAlignment="Center"                HorizontalContentAlignment="Center" FontSize="15"/>           <!-- 'Down' 버튼 -->           <RepeatButton Height ="25" Width = "25" Name ="repeatRemoveValueButton"                Delay ="200" Interval ="1"                Click ="repeatRemoveValueButton\_Click" Content = "-"/>      </StackPanel> </Window> |

|  |
| --- |
| public partial class MainWindow : System.Windows.Window {     private int currValue = 0;     public MainWindow()     {         InitializeComponent();         lblCurrentValue.Content = currValue;     }     protected void repeatAddValueButton\_Click(object sender, RoutedEventArgs e)     {         // Label에 1 더하기          currValue++;         lblCurrentValue.Content = currValue;     }     protected void repeatRemoveValueButton\_Click(object sender, RoutedEventArgs e)     {         // Label에 1 빼기         currValue--;         lblCurrentValue.Content = currValue;     } } |

사용자가 RepeatButton을 클릭하게 되면 우리는 currValue 변수가 증가하고 감소하는 것을 볼 수 있을 것이다. 그리고 Label 속성의 Contetn를 설정하는 것을 볼 수 있을 것이다. 다음 [그림29-6]은 스핀 버튼 UI처럼 동작되는 버튼을 보여주고 있다.  
  
  
[그림29-6]스핀 버튼처럼 동작되는 RepeatButton  
  
  
**CheckBox와 RadioButton 다루기**  
  
앞에서 언급했듯이 CheckBox는 ToggleButton을 상속받기 때문에 둘은 “is-a”의 관계(명확한 부모와 자식의 관계)를 가지게 된다. 버튼과 체크박스의 UI가 많이 다르기 때문에 이 관계가 조금 이상해 보일 수 있지만 CheckBox 클래스는 버튼처럼 마우스와 키보드 입력을 통해서 클릭할 수 있다. 이러한 비슷한 점들이 있기 때문에 CheckBox 클래스는 ToggleButton의 다양한 Virtual 멤버들을 재정의 하였고 UI를 체크박스처럼 보이게 수정하였다. 만약 <CheckBox>를 다음과 같이 정의한다면 [그림29-7]과 같은 화면을 볼 수 있을 것이다.

|  |
| --- |
| <StackPanel>     <!-- CheckBox types -->     <CheckBox Name ="checkInfo" >Send me more information</CheckBox>     <CheckBox Name ="checkPhoneContact" >Contact me over the phone</CheckBox> </StackPanel> |

http://www.hoons.kr/FileUpload/포맷변환_29-7.png   
[그림29-7] 간단한 CheckBox 컨트롤  
  
  
RadioButton또한 마찬가지로 ToggleButton과 “is-a”의 관계를 가지는 컨트롤이지만 CheckBox와는 다르게 같은 컨테이너(패널, 그리드)에 있는 모든 RadoButton들은 추가적인 작업 없이도 단 하나만 선택될 수 있게 설정된다. 예를 들어 다음과 같은 XAML이 있다고 가정해보자.

|  |
| --- |
| <StackPanel>     <!-- 음악 플레이어들의 선택 -->     <Label FontSize = "15" Content = "Select Your Music Media"/>     <RadioButton>CD Player</RadioButton>     <RadioButton>MP3 Player</RadioButton>     <RadioButton>8-Track</RadioButton>          <!-- 색의 선택 -->     <Label FontSize = "15" Content = "Select Your Color Choice"/>     <RadioButton>Red</RadioButton>     <RadioButton>Green</RadioButton>     <RadioButton>Blue</RadioButton> </StackPanel> |

만약 이 XAML을 가지고 테스트 해보면 이 여섯 개의 XAML 중에 단 한 개만 선택되는 것을 볼 수 있을 것이다. 만약 2개의 값을 선택할 수 있게 분리하고 싶다고 그룹으로 묶어 주어야 한다.   
  
  
**그룹의 지정**  
단 하나의 패널 위에 여러 개의 RadioButton 컨트롤들을 이용해야 하고 또한 각각의 그룹별로 구분되어야 한다면 GroupName 속성을 이용해서 그룹을 지정해 줄 수 있다. 다음 코드를 살펴보자.

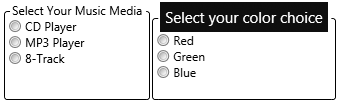
|  |
| --- |
| <StackPanel>     <!-- 음악 플레이어 그룹 -->     <Label FontSize = "15" Content = "Select Your Music Media"/>     <RadioButton GroupName = "Music" >CD Player</RadioButton>     <RadioButton GroupName = "Music" >MP3 Player</RadioButton>     <RadioButton GroupName = "Music" >8-Track</RadioButton>      <!-- 색의 그룹 -->     <Label FontSize = "15" Content = "Select Your Color Choice"/>     <RadioButton GroupName = "Color">Red</RadioButton>     <RadioButton GroupName = "Color">Green</RadioButton>     <RadioButton GroupName = "Color">Blue</RadioButton> </StackPanel> |

이처럼 컨트롤들이 물리적으로 같은 패널에 있다 하더라도 각각의 그룹별로 동작하게 하는 것이 가능하다.

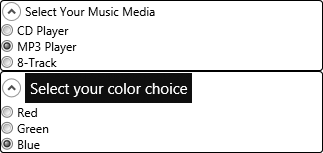
|  |
| --- |
| 기본적으로 패널 안에 있는 모든 RadioButton들은 GroupName 값을 가지지 않고 하나의 그룹으로 동작할 것이다. 그렇기 때문에 앞의 예제에서 하나의 그룹에는 그룹 이름을 지정해주지 않아도 똑같이 동작하게 된다. |

**GroupBox의 설정**  
라이오 버튼이나 체크박스 리스트들을 추가할 경우 일반적으로 눈으로 확인할 수 있는 특정 패널 안에 배치시켜 그룹처럼 동작하게 할 것이다. 이렇게 구성하기 가장 좋은 컨트롤이 바로 GroupBox라는 컨트롤이다. Header 속성은 System.Object로 어떤 개체든 할당해 줄 수 있다. 예를 들어 간단한 문자열이나 도형 버튼과 같은 컨트롤들을 Header에 추가할 수 있다. 그럼 두 개의 GroupBox를 선언하고 이전에 살펴봤던 RadioButton들을 추가해 보도록 하자.

|  |
| --- |
| <StackPanel>     <GroupBox Header = "Select Your Music Media" BorderBrush ="Black">         <StackPanel>             <RadioButton GroupName = "Music" >CD Player</RadioButton>             <RadioButton GroupName = "Music" >MP3 Player</RadioButton>             <RadioButton GroupName = "Music" >8-Track</RadioButton>         </StackPanel>     </GroupBox>     <GroupBox BorderBrush ="Black">         <GroupBox.Header>             <Label Background = "Blue" Foreground = "White"                 FontSize = "15" Content = "Select your color choice"/>         </GroupBox.Header>         <StackPanel>             <RadioButton>Red</RadioButton>             <RadioButton>Green</RadioButton>             <RadioButton>Blue</RadioButton>         </StackPanel>     </GroupBox> </StackPanel> |

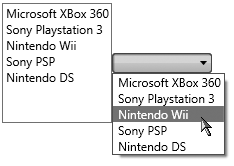
이 XAML을 출력해 보면 다음과 같이 구성되는 것을 볼 수 있을 것이다.  
  
   
[그림29-8] RadioButton 컨트롤에 적용한 GroupBox  
  
  
**Expander의 설정**  
일반적으로 많이 사용되어 왔던 그룹박스에 추가로 WPF는 특정 UI영역을 숨기고 보이게 할 수 있는 새로운 컨트롤을 제공하고 있다. 이 컨트롤 이름은 Expander 컨트롤이고 ExpandDirection 속성을 이용해서 이 요소의 방향을(위, 아래, 오른쪽, 왼쪽) 지정할 수 있다. 그럼 앞에서 사용했던 GroupBox를 Expander로 변경해 보도록 하자.

|  |
| --- |
| <StackPanel>     <Expander Header = "Select Your Music Media" BorderBrush ="Black">         <StackPanel>             <RadioButton GroupName = "Music" >CD Player</RadioButton>             <RadioButton GroupName = "Music" >MP3 Player</RadioButton>             <RadioButton GroupName = "Music" >8-Track</RadioButton>         </StackPanel>     </Expander>     <Expander BorderBrush ="Black">         <Expander.Header>             <Label Background = "Blue" Foreground = "White"                 FontSize = "15" Content = "Select your color choice"/>         </Expander.Header>         <StackPanel>             <RadioButton>Red</RadioButton>             <RadioButton>Green</RadioButton>             <RadioButton>Blue</RadioButton>         </StackPanel>     </Expander > </StackPanel> |

다음 [그림29-9]는 접혀있는 Expander를 보여주고 있다.   
  
http://www.hoons.net/FileUpload/포맷변환_29-9.png   
[그림29-9] 접혀있는 Expander  
  
다음 [그림29-10]은 확장되어 있는 Expander를 보여주고 있다.   
   
  
[그림29-10] 확장된 Expander  
  
  
**ListBox와 ComboBox 컨트롤 다루기**  
WPF 클래스는 ListBox와 ComboBox처럼 특정 항목들을 그룹으로 묶어 보여줄 수 있는 컨트롤들을 제공하고 있다. 이 두 컨트롤들은 ItemsControl 추상 클래스로부터 파생되었다. 여기서 중요한 것은 이 부모 클래스는 Items라는 속성을 정의하고 있다는 것이고 이 속성은 자식항목을 가지고 있는 ItemCollection 객체를 반환 한다는 것이다. ItemCollection 클래스는 System.Object 타입의 자식들을 가지고 있을 수 있기 때문에 어떤 자식이든 가지고 있을 수 있다. 만약 마크업 코드를 이용해서 간단한 문자열을 ItemsControl에 추가하고 싶다면 <ListBoxItem>를 이용해서 추가할 수 있을 것이다. 예를 들어 다음과 같이 XAML 코드를 작성할 수 있다.

|  |
| --- |
| <!-- 간단한 LIstBox--> <ListBox Name = "lstVideoGameConsoles">     <ListBoxItem>Microsoft XBox 360</ListBoxItem>     <ListBoxItem>Sony Playstation 3</ListBoxItem>     <ListBoxItem>Nintendo Wii</ListBoxItem>     <ListBoxItem>Sony PSP</ListBoxItem>     <ListBoxItem>Nintendo DS</ListBoxItem> </ListBox>   <!-- 간단한 ComboBox --> <ComboBox Name = "comboVideoGameConsoles">     <ListBoxItem>Microsoft XBox 360</ListBoxItem>     <ListBoxItem>Sony Playstation 3</ListBoxItem>     <ListBoxItem>Nintendo Wii</ListBoxItem>     <ListBoxItem>Sony PSP</ListBoxItem>     <ListBoxItem>Nintendo DS</ListBoxItem> </ComboBox> |

|  |
| --- |
| ComboBox 컨트롤은 <ListBoxItem>대신 <ComboBoxItem> 라는 요소를 이용할 수 있다. 이 요소를 이용하면 ListBoxItem 에서는 지원되지 않는 IsHighlighted 속성을 지원해주고 있고 이 속성을 이용해서 특정 효과를 주는 것이 가능하다. |

이 컨트롤은 예상한 것처럼 다음 [그림29-11]처럼 보여지게 된다.  
  
   
[그림29-11] 간단한 ListBox와 ComboBox 컨트롤  
  
  
**프로그램에서 리스트 컨트롤 다루기**  
  
보통 리스트 컨트롤의 데이터들은 실행하기 전까지 알 수 없을 때가 많다. 예를 들어 어떤 DB에서 값을 읽어 리스트를 작성할 수도 있고 또한 WCF를 이용하거나 아니면 외부 파일을 읽어서 리스트를 작성하기도 할 것이다. 이렇게 C#코드를 이용해서 ListBox나 ComboBox의 리스트를 할당해주어야 한다면 ItemCollection에서 제공하는 Add(), Remove()와 같은 여러 메서드들을 이용해서 간단하게 구현할 수 있다. 그럼 비주얼 스튜디오에서 ListControls라는 WPF 응용 프로그램 프로젝트를 생성하고 lstVideoGameConsole이라는 ListBox를 정의해 보도록 하자.

|  |
| --- |
| <Window x:Class="ListControls.MainWindow"     xmlns="[**http://schemas.microsoft.com/winfx/2006/xaml/presentation**](http://schemas.microsoft.com/winfx/2006/xaml/presentation)"     xmlns:x="[**http://schemas.microsoft.com/winfx/2006/xaml**](http://schemas.microsoft.com/winfx/2006/xaml)"     Title="ListControls" Height="300" Width="300" >      <StackPanel>         <!-- 코드를 통해서 리스트를 채울 것이다. -->         <ListBox Name = "lstVideoGameConsoles">         </ListBox>     </StackPanel> </Window> |

그리고 C#코드에서는 다음과 같은 코드를 작성해 보도록 하자.

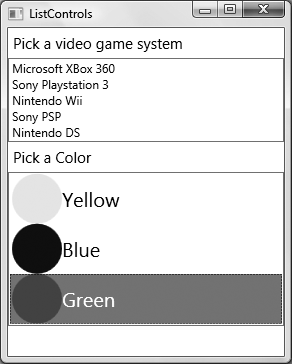
|  |
| --- |
| public partial class MainWindow : System.Windows.Window {     public MainWindow()     {         InitializeComponent();         FillListBox();     }     private void FillListBox()     {         // 항목 추가         lstVideoGameConsoles.Items.Add("Microsoft XBox 360");         lstVideoGameConsoles.Items.Add("Sony Playstation 3");         lstVideoGameConsoles.Items.Add("Nintendo Wii");         lstVideoGameConsoles.Items.Add("Sony PSP");         lstVideoGameConsoles.Items.Add("Nintendo DS");     } } |

앞에 XAML에서 항목을 추가할 때는 ListBoxItem 이라는 속성을 이용해서 항목을 추가하였지만 코드에서는 string 타입의 값을 Add() 메서드를 이용해서 바로 추가하였다. XAML에서 특정 항목을 추가한다면 [**http://schemas.microsoft.com/winfx/2006/xaml/presentation**](http://schemas.microsoft.com/winfx/2006/xaml/presentation) XML 네임스페이스에 정의되어 있는 <ListBoxItem> 타입을 이용하면 더 편리하게 이용할 수 있다는 것이 결론이다.   
  
실제로 <ListBoxItem>각각은 ToString()이 호출되기 때문에 동일한 결과를 볼 수 있고 만약 XAML에서 ListBox의 항목을 System.String을 이용해서 추가하고 싶다면 mscorlib.dll이라는 새로운 네임스페이스를 추가해주어야 한다.

|  |
| --- |
| <StackPanel xmlns:CorLib = "clr-namespace:System;assembly=mscorlib">     <ListBox Name = "lstVideoGameConsoles">         <CorLib:String>Microsoft XBox 360</CorLib:String>         <CorLib:String>Sony Playstation 3</CorLib:String>         <CorLib:String>Nintendo Wii</CorLib:String>         <CorLib:String>Sony PSP</CorLib:String>         <CorLib:String>Nintendo DS</CorLib:String>     </ListBox> </StackPanel> |

반대로 코드에서 ItemsControl에서 파생된 ListBoxItem을 이용해서 항목을 추가하고 싶을 수 있을 것이다. 하지만 실제로는 이렇게 작성해도 크게 달라지는 것이 없다는 것을 알아두자.  
  
  
**임의적인 개체 추가하기**  
ListBox와 ComboBox는 ContentControl이라는 클래스를 상속받기 때문에 간단한 문자열뿐만 아니라 다양한 데이터들을 가질 수 있다. 예를 들어 2D 그래픽 개체와 설명을 위한 Label을 가지고 있는 ListBox 컨트롤을 구현해 보도록 하자.

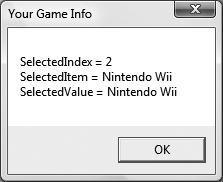
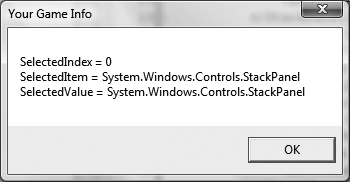
|  |
| --- |
| <StackPanel>  <!-- ListBox의C Content -->  <ListBox Name = "lstColors">  <StackPanel Orientation ="Horizontal">  <Ellipse Fill ="Yellow" Height ="50" Width ="50"/>  <Label FontSize ="20" HorizontalAlignment="Center"  VerticalAlignment="Center">Yellow</Label>  </StackPanel>  <StackPanel Orientation ="Horizontal">  <Ellipse Fill ="Blue" Height ="50" Width ="50"/>  <Label FontSize ="20" HorizontalAlignment="Center"  VerticalAlignment="Center">Blue</Label>  </StackPanel>  <StackPanel Orientation ="Horizontal">  <Ellipse Fill ="Green" Height ="50" Width ="50"/>  <Label FontSize ="20" HorizontalAlignment="Center"  VerticalAlignment="Center">Green</Label>  </StackPanel>  </ListBox>  </StackPanel> |

다음 [그림29-12]는 현재 리스트를 포함하고 있는 ListBox를 보여주고 있다.  
   
  
  
[그림29-12] ItemsControl에서 파생된 컨트롤은 어떤 내용이든 포함할 수 있다.  
  
  
**선택한 항목 가져오기**  
  
ListBox나 ComboBox를 추가한 다음에 분명 사용자가 선택한 값을 어떻게 가져와야 할지 궁금하게 될 것이다. 이 값을 가져오는 방법은 세가지로 정리해볼 수 있다. 먼저 선택된 아이템의 Index를 조회하고 싶다면 SelectedIndex 속성을 이용해서 현재 선택된 Index의 값을 가져올 수 있을 것이다. 만약 선택된 항목이 참조하고 있는 객체를 가져오고 싶다면 SelectedItem 속성을 이용해서 그 객체를 접근할 수 있을 것이다. 마지막으로 SelectedValue를 이용한다면 현재 선택된 값을 가져올 수 있다. 이 정도로 굉장히 간단하다. 그럼 각각의 속성들이 어떻게 동작되는지 확인해보기 위해서 다음과 같이 2개의 새로운 버튼을 만들고 Click 이벤트를 각각 설정해 주도록 하자.

|  |
| --- |
| <!-- 선택된 항목을 가져오는 버튼 --> <Button Name ="btnGetGameSystem" Click ="btnGetGameSystem\_Click">   Get Video Game System </Button> <Button Name ="btnGetColor" Click ="btnGetColor\_Click">   Get Color </Button> |

btnGetGameSystem의 Click 이벤트가 발생하면 lstVideoGameConsoles 객체의 SelectedIndex와 SelectedItem, SelectedValue 속성들이 반환하는 값을 메시지 박스로 보여주는 코드를 작성해 보도록 하자.

|  |
| --- |
| protected void btnGetGameSystem\_Click(object sender, RoutedEventArgs args)  {  string data = string.Empty;  data += string.Format("SelectedIndex = {0}\n",  lstVideoGameConsoles.SelectedIndex);  data += string.Format("SelectedItem = {0}\n",  lstVideoGameConsoles.SelectedItem);  data += string.Format("SelectedValue = {0}\n",  lstVideoGameConsoles.SelectedValue);  MessageBox.Show(data, "Your Game Info");  } |

게임 리스트 중에서 만약 “Nintendo Wii”를 선택한 후에 버튼을 클릭하게 되면 다음 [그림29-13]과 같이 메시지 박스가 출력되는 것을 볼 수 있을 것이다.  
  
   
[그림29-13] 선택된 값 찾기  
  
하지만 선택된 색은 어떻게 알아올 것인가?  
  
**Content에서 선택한 항목 가져오기**  
  
btnGetColor 버튼의 Click 이벤트로 동작되는 btnGetColor\_Click() 메서드 또한 앞의 예제와 처럼 작성하여 lstColors의 선택된 인덱스와 값을 가져와 보도록 하자. 만약 lstColors의 첫 번째 항목을 선택한 후에 버튼을 클릭하면 다음 [그림29-14]와 같은 메시지 창이 출력되는 것을 볼 수 있을 것이다.   
   
  
[그림29-14] 선택된 값이 StackPanel?  
  
이렇게 값을 가져오는 이유는 실제로 lstColors 객체는 세 개의 StackPanel 객체를 가지고 있는 것이고 이 패널들은 각각 자식으로 값을 가지고 있는 것이다. 그렇기 때문에 SelectedItem과 SelectedValue는 StackPanel 컨트롤을 ToString()한 값이 출력되는 것이다.   
  
SelectedIndex에서 반환한 값을 이용해서 선택된 항목을 간단하게 찾을 수도 있지만 StackPanel의 컬렉션을 가져온 후에 StackPanel이 내부적으로 가지고 있는 자식 컬렉션을 이용해서 Label의 Content 값을 직접 가져올 수도 있다. 다음 코드를 살펴보자.

|  |
| --- |
| protected void btnGetColor\_Clicked(object sender, RoutedEventArgs args) {     // 선택된 StackPanel을 찾은 후에 Label을 가져온다.     StackPanel selectedStack =     (StackPanel)lstColors.Items[lstColors.SelectedIndex];     string color = ((Label)(selectedStack.Children[1])).Content.ToString();     string data = string.Empty;     data += string.Format("SelectedIndex = {0}\n", lstColors.SelectedIndex);     data += string.Format("Color = {0}", color);     MessageBox.Show(data, "Your Game Info"); } |

이러한 코드는 Label의 위치가 하드코딩 되어 있기 때문에 만약 StackPanel 안에 다른 컨트롤들이 추가 된다면 오류가 발생할 확률이 높다. 그 대안으로 StackPanel의 Tag라는 속성을 각각 정의해 주어도 된다.

|  |
| --- |
| <ListBox Name = "lstColors">  <StackPanel Orientation ="Horizontal" Tag ="Yellow">  ...  </StackPanel>  <StackPanel Orientation ="Horizontal" Tag ="Blue">  ...  </StackPanel>  <StackPanel Orientation ="Horizontal" Tag ="Green">  ...  </StackPanel>  </ListBox> |

이 방법을 이용하면 Tag값만 프로그램에서 가져오면 되기 때문에 우리는 간단하게 처리할 수 있다. 다음 코드를 살펴보자.

|  |
| --- |
| protected void btnGetColor\_Clicked(object sender, RoutedEventArgs args)  {  string data = string.Empty;  data += string.Format("SelectedIndex = {0}\n", lstColors.SelectedIndex);  data += string.Format("SelectedItem = {0}\n", lstColors.SelectedItem);  data += string.Format("SelectedValue = {0}",  (lstColors.Items[lstColors.SelectedIndex] as StackPanel).Tag);  MessageBox.Show(data, "Your Color Info");  } |

이 방법이 첫 번째 방법보다 약간은 더 안전하긴 하지만 또 다른 방법은 바로 데이터 템플릿을 이용해서 복잡한 컨트롤들의 값을 가져오는 방법이 있다. 이 방법을 이용하기 위해서는 먼저 데이터 바인딩을 이해하고 있어야 한다. 데이터 바인딩에 대해서는 뒷부분에서 살펴볼 것이다.   
  
  
**Text 컨트롤 다루기**  
  
WPF는 텍스트 값을 입력 받는 몇 가지 UI요소들을 제공하고 있다. 여기서는 가장 기본적으로 TextBox 와 PasswordBox 컨트롤을 살펴볼 것이다. 그럼 TextControls라는 WPF 응용프로그램 프로젝트를 생성하도록 하자.  
  
**TextBox 컨트롤**  
  
다른 TextBox와 마찬가지로 WPF에서 제공하는 TextBox는 한 줄만 입력할 수 있게 설정할 수 있고 아니면 AcceptReturn 속성을 true로 설정하여 여러 라인을 입력받을 수 있게 설정할 수 있다. TextBox의 정보는 언제나 문자열이기 때문에 Content는 언제나 String 타입으로 Text 속성을 이용해서 반환하게 된다.

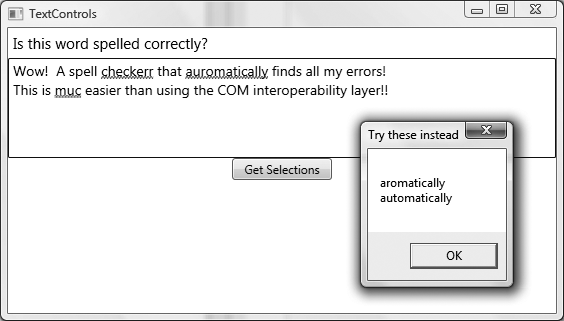
|  |
| --- |
| <TextBox Name ="txtData" Text = "Hello!" BorderBrush ="Blue" Width ="100"/> |

WPF의 TextBox 컨트롤의 특별한 기능은 SpellCheck.IsEnabled를 true로 설정하게 되면 특정 문자열의 스펠링을 체크해주게 할 수 있다는 것이다. 이렇게 설정하면 Microsoft Office에서처럼 잘못된 스펠링에 대해서 밑줄이 그어지는 것을 볼 수 있다. 더 놀라운 것은 현재 TextBox에서 잘못된 단어로 사용된 단어들의 리스트를 가져올 수 있다는 것이다.  
  
그럼 다음과 같이 Label, TextBox, Button과 같은 컨트롤들을 추가하여 XAML 수정해 보도록 하자.

|  |
| --- |
| <Window x:Class="TextControls.MainWindow" xmlns="[**http://schemas.microsoft.com/winfx/2006/xaml/presentation**](http://schemas.microsoft.com/winfx/2006/xaml/presentation)" xmlns:x="[**http://schemas.microsoft.com/winfx/2006/xaml**](http://schemas.microsoft.com/winfx/2006/xaml)" Title="TextControls" Height="204" Width="292" >   <StackPanel>     <Label FontSize ="15">Is this word spelled correctly?</Label>     <TextBox SpellCheck.IsEnabled ="True" AcceptsReturn ="True"     Name ="txtData" FontSize ="12"     BorderBrush ="Blue" Height ="100">     </TextBox>     <Button Name ="btnOK" Content ="Get Selections"     Width = "100" Click ="btnOK\_Click"/>   </StackPanel> </Window> |

이렇게 작성하고 실행해보면 잘못된 단어를 넣게 되면 에러들이 표시되는 것을 볼 수 있을 것이다. 그럼 버튼을 클릭했을 때 현재 잘못 쓰여진 단어를 가져오는 프로그램을 작성해 보도록 하자.

|  |
| --- |
| protected void btnOK\_Click(object sender, RoutedEventArgs args) {     string spellingHints = string.Empty;     // 현재 잘못된 단어들의 리스트를 가져온다.      SpellingError error = txtData.GetSpellingError(txtData.CaretIndex);     if (error != null)     {         // 하나의 문자열로 만든다.          foreach (string s in error.Suggestions)         {             spellingHints += string.Format("{0}\n", s);         }         // 잘못된 단어들을 보여준다.         MessageBox.Show(spellingHints, "Try these instead");     } } |

보는 것과 같이 이 코드는 매우 간단하다. 먼저 우리는 CaretIndex 속성을 이용해서 잘못된 단어들을 받아오게 된다. 그리고 만약 그 단어가 존재한다면 루프를 돌면서 그 값을 문자열로 연결한 후에 MessageBox.Show()를 이용해서 단어들을 보여주게 된다. 다음 [그림29-15]는 이렇게 “auromatically.”라는 단어가 잘못되었다고 메시지가 뜨는 것을 보여주고 있다.  
   
  
[그림29-15] 스펠링 체크 프로그램  
  
  
**PasswordBox 컨트롤 다루기**  
  
이 PasswordBox 컨트롤은 크게 특별한 것 없이 사용자가 입력하는 값을 보호하기 위해서 다른 문자로 대체하여 보여주게 된다. 기본적으로 대체되는 문자는 “\*”을 이용하게 되지만 이 값은 PasswordChar 속성을 이용해서 원하는 값대로 수정할 수 있다. 그리고 사용자가 입력한 값을 확인하고 싶다면 Password 속성을 이용해서 값을 가져올 수 있다. 그럼 앞에서 만든 프로그램에서 비밀번호를 누른 다음에 동작이 되는 로직을 추가해 보도록 하겠다. 먼저 <StackPanel> 안에 <StackPanel>을 추가한 후에 PasswordBox와 Button을 추가하여 수평으로 위치시키도록 하자.

|  |
| --- |
| <StackPanel>   <Label FontSize ="15">Is this word spelled correctly?</Label>   <TextBox SpellCheck.IsEnabled ="True" AcceptsReturn ="True"   Name ="txtFavoriteColor" FontSize ="14"   BorderBrush ="Blue" Height ="100">   </TextBox>   <StackPanel Orientation ="Horizontal">     <PasswordBox Name ="pwdText" BorderBrush ="Black" Width ="100"></PasswordBox>     <Button Name ="btnOK" Content ="Get Selections"     Width = "100" Click ="btnOK\_Click"/>   </StackPanel> </StackPanel> |

여기서 Button이 클릭되었을 때 문자열을 하드코딩 해서 비밀번호가 올바른지 확인하는 CheckPassword() 메서드를 호출한 후에 올바른 비밀번호라면 스펠링을 확인하도록 로직을 추가해 보도록 하자.

|  |
| --- |
| public partial class MainWindow : System.Windows.Window {     ....     protected void btnOK\_Click(object sender, RoutedEventArgs args)     {         if (CheckPassword())         {             // 이전에 작성했던 잘못된 스펠링을 검사하는 코드         }         else             MessageBox.Show("Security error!!");     }     private bool CheckPassword()     {         if (pwdText.Password == "Chucky")             return true;         else             return false;     } } |

TextBox와 PasswordBox에 만약 그래픽 개체나 특정 개체를 포함시킬 수 있다. 또한 WPF는 RichTextBox를 제공해주고 있기 때문에 문자열 형식에 특정 포맷을 지정할 수 있다. 그리고 WPF는 System.Windows.Documents 네임스페이스를 통해서 문서 전체를 읽어서 보여줄 수 있는 API또한 제공하고 있다.   
  
WPF는 Flow 문서를 만들기 위한 여러 가지 API들을 제공해주고 있기 때문에 텍스트뿐만 아니라 주석, 테이블과 같은 요소들을 추가할 수 있을 것이다. 이번 책에서는 RichTextBox나 Flow 문서들에 대한 API를 다루지는 않을 것이지만 만약 이 내용을 살펴보고 싶다면 .NET Framework 3.5 SDK에서 제공하고 있는 가이드를 살펴 본다면 보다 쉽게 이해할 수 있을 것이다.  
  
지금까지 WPF의 기본적인 컨트롤들에 대해서 살펴 보았다. 뒷부분에서는 메뉴, 상태바, 툴바와 같은 컨트롤을 살펴볼 것이지만 먼저 UI들을 어떻게 배치시켜야 하는지 이해하기 위해서 패널에 대해서 살펴보도록 하겠다.